Report Assignment 6

December 5, 2015

Purpose/Description of Assignment

The purpose of this assignment was to learn about data structures. Namely, understand how the Graph data structure operates and how to implement Kruskal's minimum spanning tree algorithm. In this assignment the Graph data structure was implemented using the DisjointSet class created in assignment 5 along with doubly linked lists. An adjacency list was used to store the vertecies. The member functions of Graph are: BuildGraph(), insertEdge(), getWeight() , sortEdge(), and MSTAlgo(). The last function, MSTAlgo(), is an implementation of Kruskal's minimum spanning tree algorithm.

Algorithms & Data Strucutres

The structure of the Graph class is simple. First there are two structures: an Edge and a Vertex. The Edge structure has the indicies of two verticies and its weight. The Vertex structure has an index and an Edge. The Graph class uses both of these structures, along with a class name DisjointSet which is used to implement Kruskal's minimal spanning tree algorithm. The Graph class uses an AdjacencyList (vector of doubly linked lists) to store the verticies. An EdgeList (a vector of edges) stores the edges. It also has another vector of edges to store the minimum spanning tree. The Graph class has five main member functions: buildGraph(), insertEdge(), getWeight(), sortEdge(), MSTAlgo().

BuildGraph

The buildGraph() member function takes no parameters. It simply creates a new DListNode of verticies for every vertex read in from a file and adds them into the AdjacenyList. The time complexity for this algorithm is O(n). Since creating a DListNode of one verticie takes constant time, O(1), but creating n DListNodes each with one vertex will take n*O(1) which is O(n).

InsertEdge

The member function InsertEdge acepts three parameters: an index i, an index j, and a double weight. The functions creates an edges with the information provided and adds it to the data member named EdgeList. It then creates two verticies, one with index i, and the other with index j. It then assigns the member data member Edge of the verticies to point to the created edge. Lastly, both of these verticies are inserted into there appropriate slots in the adjaceny matrix. The time complexity for this algorithm is O(1). It takes constant time to create the edge, two verticies, and to link the edge with the verticies. Inserting into the list also takes O(1) since we already know where we want to insert the edge. Hence the overal running time is O(1).

GetWeight

The member function getWeight() takes two integers as parameters. The two integers represent verticies. The function first creates an a node that points to Adjacency[i], ie the node is point to vertex i. The node then iterates through the nodes conected to vertex i, until it finds vertex j. Finally it returns the weight of the verticie connecting vetex i to vertex j. Since the function has to iterate through the doubly linked list, the runtime is O(n).

SortEdge

In order to sort the vectors, the standard library function was used. At first the sortEdge function was implemented using insertion sort since it was the easiest and quickest to implement. However the desired running time was not achieved using this sort, so the standard libary sort was used. A comparision function was created in order to designate the std sort function to compare using weights. The total running time for this function is$O(nlogn)$.

MSTAlgo

The member function MSTAlgo() is an an implementation of Kruskal's minimum spanning tree algorithm. This function makes use of the disjoint set class created in programming assignment 5. The function creates an instantance of the DisjointSet<Vertex>. It makes a set for every vertex in AdjacencyList. It then sorts the edges in EdgeList using the sortEdge() function described above. The smallest edge in EdgeList is chosen and checked to see if the two verticies creating that edge are in different sets. If they are, then they are combined into one set using the Union function. The edge is also added into the MST vertex because it is the smallest edge that connects to verticies. When edgeList is completley traversed, the function ends and MST now has the minimum spanning set. The function returns the sum of the edges in the minimum spanning set. The Algorithm makes use of union by rank and path compression to make a set in O(1), find a set in O(1) and union a set in O(log n). The total running time for this algorithm is $O((E + V)logV)$

Program Organization

The program is split up into separate files for ease of use. There is a Graph class which contains two structures: a Vertex and an Edge. The Graph class is split up even further. One file, Graph.cpp, contains the definitions, while Graph.h contains the declarations. There is a class, DListNode, which is simply a doubly linked list and is use to to create an Adjancency list for the graph. This class has all its implementations in one file because it is templated and cannot be split up. The final class that was used was the disjointSet class. This class was utilized to implement Kruskal's minimum spanning tree algorithm. Like the DListNode class, this class is also generic so all the declarations and definitions were done in the header file.

Instructions to Compile and Run

To compile the program you must first be in the directory Lopez-Guillermo-A6. Once in this directory, type the command "make all" into the unix terminal. This will compile the program. Now in order to run the program you must supply the main.cpp with a .mat file. For example, currently there is a test1.mat file in the directory. The command to run the program with that file would be "./main test1.mat". The file passed to main must contain the number of verticies and edges at the top of the file. The following lines must contain the verticies with weights. To mark the end of a line -1 must be entered. After starting the program, it does not ask for any additional user input.

Input and Output Specifications

The input the program takes is a .mat file contaning information about the graph. The first line of the file has the number of vertices followed by a space followed by the number of edges. The next lines are as follows: the line represents a vertex and the numbers are: a vertex followed by its weight. The line ends with the sentinel -1. An example is shown below.

```
4 4
1 9 2 3 3 5 -1
0 9 3 2 -1
0 3 -1
0 5 1 2 -1
```

This means data means: there are 4 verticies and 4 edges. vertex zero is connected to vertex 1 by a weight of 9. Vertex 0 is connected to vertex 2 by a weight of 3. Vertex 0 is connected to vertex 3 by a weight of 5. -1 indicates that we are done with that line. The second line reads: vertex 1 is connected to vertex 0 by a weight of 9. Vertex 1 is connected to 3 by a weight of 2. The end of the line is reached. This continues for the remaining verticies. The output of the program is an adjacey matrix of the graph followed by its minimum spanning tree weight and the minimum spanning tree. Example output is shown below.

```
The Adjacency Matrix of the Graph is:
0 9 3 5
9 0 0 2
3 0 0 0
```

```
5 2 0 0
The total value of the Minimum Spanning Tree is: 10
The Minimum Spanning Tree is:
Node Node Weight
1 3 2
0 2 3
0 3 5
```

Logical Exceptions

The program will catch several logical exceptions. For instance an exception will be thrown when the number of edges called to build a graph is negative. An exception is also thrown if the number of edges inserted does not match what was specified in the input. Program will crash if letters are inputed instead of numbers in the .mat file. Also if edges are directed, that is there is an edge to vertex a from b, but not from b to a, then the program will throw a segmentation fault error.

C++ Generic/OO Programming Features

Templates were used in both the implementation of DListNode and DisjointSet. This was done so that the program was not confined to one data type. The program was also implemented keeping encapsulation in mind. All data members are kept private and can only be accessed by using member functions. This protects the integrity of the data. The Graph data structure is not templated, but since it is using DListNode, DisjointSet, and vectors (all classes which are generic) it has the cababilities to be used with other data types.

Tests

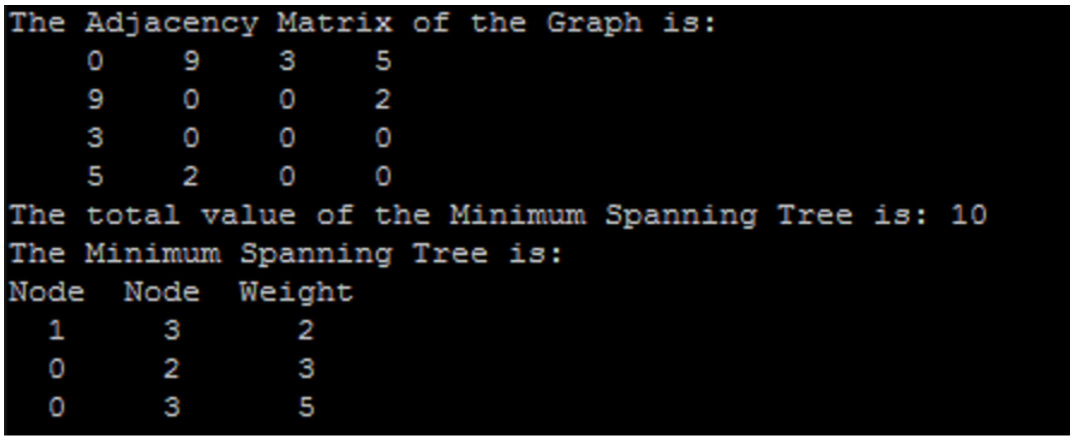
Results of testing is shown below. Test 1 and 2 show correct ouput with input specified above the picture. Test 3 shows incorrect output with input shown above the picture.

Test 1

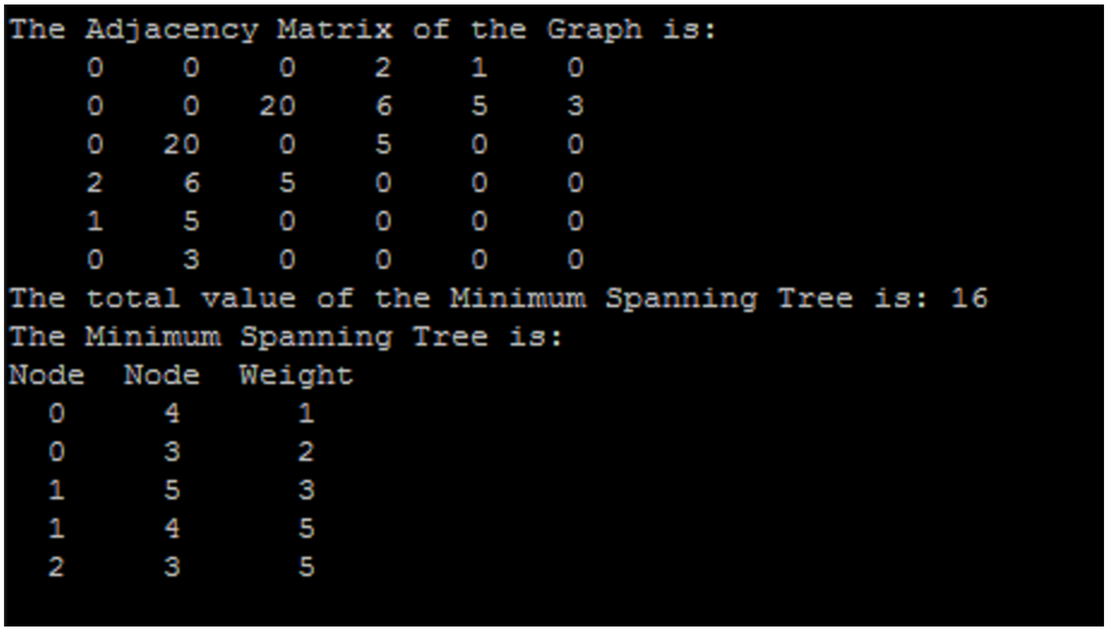
Input:

```
4 4
1 9 2 3 3 5 -1
0 9 3 2 -1
0 3 -1
0 5 1 2 -1
```

Output:


```
The Adjacency Matrix of the Graph is:
    0    9    3    5
    9    0    0    2
    3    0    0    0
    5    2    0    0
The total value of the Minimum Spanning Tree is: 10
The Minimum Spanning Tree is:
Node  Node  Weight
  1     3      2
  0     2      3
  0     3      5
```

Test 2
Input:
6 7
3 2 4 1 -1
2 20 3 6 4 5 5 3 -1
1 20 3 5 -1
0 2 1 6 2 5 -1
0 1 1 5 -1
1 3 -1


```
The Adjacency Matrix of the Graph is:
    0    0    0    2    1    0
    0    0   20    6    5    3
    0   20    0    5    0    0
    2    6    5    0    0    0
    1    5    0    0    0    0
    0    3    0    0    0    0
The total value of the Minimum Spanning Tree is: 16
The Minimum Spanning Tree is:
Node  Node  Weight
  0     4      1
  0     3      2
  1     5      3
  1     4      5
  2     3      5
```

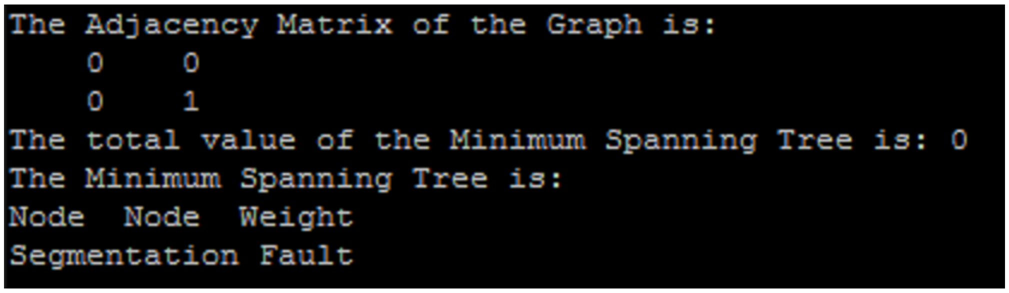
Test 3: Program crashes because edges are not specified correctly in the .mat file.

Input:

2 2

-1 1

1 -1



```
The Adjacency Matrix of the Graph is:
    0    0
    0    1
The total value of the Minimum Spanning Tree is: 0
The Minimum Spanning Tree is:
Node  Node  Weight
Segmentation Fault
```